الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic and Republic of Algeria

Ministry of Higher Education and Scientific Research

National School of Cyber Security

وزارة التعليم العالي والبحث العلمي

المدرسة الوطنية العليا في الأمن السيبراني

Second Semester Project

Level: 1st Year **Material:** Algorithms and Dynamic Data Structures

1. Project description

This project focuses on implementing fundamental data structures, such as chars, strings, linked lists, stacks, queues, recursion, and trees, and analyzing their complexity. It mainly consists of two main parts.

2. First part

The first part consists of developing the necessary functions and procedures to efficiently manage a security logs system.

2.1. Module 1: Linked Linear Lists (linkedLists.c/linkedLists.h) (14 points)

- Insert Log Entry (at the beginning, at the end, at a specific position).
- Delete Log Entry (by ID, by timestamp, delete first/last).
- Search Log Entry (by ID, by keyword, by timestamp).
- Sort Logs (by date, by severity level).
- Reverse the List (to view logs in reverse order).
- Count Total Logs (return the number of log entries).

2.2. Module 2: Bidirectional Linked Lists (bidirectionalLinkedLists.c/bidirectionalLinkedLists.h) (18 points)

- All functions from singly linked list but with bidirectional navigation.
- Move Forward and Backward through logs.
- Delete Middle Element (remove a log at a specific index).
- Merge Two Log Lists (combine logs from two sources).

2.3. Module 3: Circular Linked Lists (circularLists.c/circularLists.h) (16 points)

- All functions from singly linked list but with circular traversal.
- Implement a Fixed-Size Log Buffer (overwrite old logs automatically).
- Detect Cycles in the List (validate log data consistency).

2.4. Module 4: Queues (queues.c/queues.h) (10 points)

- Enqueue New Log Entry (insert log at the end).
- Dequeue Log Entry (remove log from the front).
- Peek (view the first log without removing it).
- Check if Queue is Empty or Full.
- Circular Queue Implementation (for efficient memory usage).

2.5. Module 5: Stacks (stacks.c/stacks.h) (06 points)

- Push New Log Entry (insert log at the top).
- Pop Log Entry (remove log from the top).
- Peek (view the top log without removing it).
- Check if Stack is Empty or Full.
- Reverse a Stack Using Recursion.

2.6. Module 6: Recursion (recursion.c/recursion.h) (07 points)

- Reverse a Linked List Using Recursion.
- Calculate Factorial and Fibonacci Using Recursion.

Academic Year: 2024/2025

Second Semester Project

Level: 1st Year **Material:** Algorithms and Dynamic Data Structures

- Find Maximum Log Entry ID Using Recursion.
- Implement Recursive Binary Search in Sorted Logs.
- Convert an Infix Expression to Postfix Using Recursion.

2.7. Module 7: Trees (trees.c/trees.h) (09 points)

- Binary Search Tree (BST) for Log Searching.
- Insert Log into BST (logs sorted by timestamp).
- Delete Log from BST.
- Search Log in BST.
- Traverse Logs in Different Orders (In-order, Pre-order, Post-order).
- Convert Linked List to BST (to improve search efficiency).
- Implement a Heap Structure for Efficient Log Management.

2.8. Module 8: Complexity Analysis (complexity.c/complexity.h) (06 points)

- Measure Execution Time for different operations (Insertion, Deletion, Search).
- Compare Performance of Different Data Structures.
- Analyze Space Complexity for different implementations.

3. Second part

The second part involves managing a dictionary of words containing synonyms and antonyms, stored in a text file. Words and their synonyms are separated by "=", while words and their antonyms are separated by "#". The user has access to a complete menu to manage the stored data, including the following functions and procedures:

3.1. Modules based on Linked lists and Queues (18 points)

- **TList *getSynWords(File *f):** this function puts all words with their synonyms into a linked list, where the node contains the words, their synonym, number of chars and vowels.
- **TList *getAntoWords(File *f):** this function puts all words with their antonyms into a linked list, where the node contains the words, their antonym, number of chars and vowels.
- **void getInfWord(TList *syn, TList *ant, char *word):** this procedure takes a word as input and returns the synonym, antonym, number of chars and vowels.
- **void getInfWord2(TList *syn, TList *ant, char *inf):** this procedure takes synonym or antonym as input and returns the word with number of chars and vowels.
- **TList *sortWord(TList *syn):** this function sorts the words alphabetically.
- **TList *sortWord2(TList *syn):** this function sorts the list of nodes in ascending order according to the number of chars.
- **TList *sortWord3(TList *syn):** this function sorts the list nodes in descending order according to the number of vowels.
- **TList *deleteWord(File *f, TList *syn, TList *ant, char *word):** this function deletes a word from the original text file and both linked lists.
- **TList *updateWord(File *f , TList *syn, TList *ant, char *word, char *syne, char *anton):** this function updates a word synonym and antonym on the original text file and also their information in both linked lists.
- **TList *similarWord(TList *syn, char *word, rate):** this function returns a list that contains all similar words with match rate great than or equal to **rate**.

Second Semester Project

Level: 1st Year **Material:** Algorithms and Dynamic Data Structures

- **TList *countWord(TList *syn, char *prt):** this function returns a list that contains all words where the string **prt** is a part of it.
- **TList *palindromWord(TList *syn):** this function returns a list containing all sorted palindrome words with their information; each word is inserted in the right place to get the sorted list.
- **TList *merge(TList *syn, TList *ant):** this function merges the two nodes from two lists into one node of a bidirectional list.
- **TList *merge2(TList *syn, TList *ant):** this function merges the two nodes of two lists into one node of a circular list.
- **TList *addWord(TList *syn, TList *ant, char *word, char *syne, char *anton):** this function adds a word with synonym and antonym into both lists, and to the text file.
- **TQueue *syllable(TList *syn):** this function sorts the words according to the number of syllables in a queue where each part is separated by empty words.
- **TQueue *prounounciation(TList *syn):** this function sorts the words into three queues according to how words are pronounced: short, long or diphthong.
- **TQueue *toQueue(TList*merged):** this function converts the list obtained from **merge** function into a **Queue**.

3.2. Modules based on Stacks (stacks.h/stacks.c) (14 points)

- **TStack *toStack(TList *merged):** this function converts the list returned by **merge** function into a stack.
- **TStack *getInfWordStack(TStack *stk, char *word):** this function takes a word as input and returns its synonym, antonym, number of chars and vowels from the stack **stk**.
- **TStack *sortWordStack(TStack *syn):** this function sorts the stack alphabetically.
- **TStack *deleteWordStack(TStack *stk, char *word):** this function deletes a word from the stack.
- **TStack *updateWordStack(TStack *stk, char *word, char *syne, char *anton):** this function updates a word synonym and antonym on the stack.
- **TQueue *stackToQueue(TStack *Stk):** this function converts a stack of function **toStack** into a sorted queue.
- **TList *StacktoList(TStack *Stk):** this function converts a stack of function **toStack** into a bidirectional sorted linked list.
- **TStack *addWordStack(TStack *stk, char *word, char *syne, char *anton):** This function adds a word along with its synonym and antonym into a sorted stack.
- **TStack *syllableStack(TStack *stk):** this function sorts the words according to the number of syllables in a stack, where each part is separated by empty words.
- **TStack *prounounciationStack(TStack *stk):** this function sorts the words into three stacks according to how they are pronounced: short, long or diphthong.
- **char *getSmallest(TStack *stk):** this function returns the smallest word in the stack.
- **void cycleSearch(TStack *Stk):** this procedure prints the Cycle word where a synonym or antonym of a word lead to another word.
- **bool isPalyndromeStack(char *word):** this function checks if a word is a palindrome using a stack.
- **TStack *StackRev(TStack *stk):** this function reverses a stack using **recursion.**

3.3. Modules based on Binary Search Tree (BST) (binarySearchTrees.h/binarySearchTrees.c) (16 points)

- **TTree *toTree(TStack *stk):** this function converts a stack to a BST.
- **TTree *fillTree(File *f):** this function returns a BST filled with words.
- **TTree *getInfWordTree(TTree *tr, char *word):** this function takes a word as input and returns its synonym, antonym, number of chars and vowels from the BST **tr**.

Second Semester Project

Level: 1st Year **Material:** Algorithms and Dynamic Data Structures

- **TStack *AddWordBST(TTree *tr, char *word, char *syne, char *anton):** this function adds a word with its synonym and antonym into a sorted BST.
- **TTree *deleteWordBST(TTree *tr, char *word):** this function deletes a word from the BST.
- **TTree *UpdateWordBST(TTree *tr, char *word, char *syne, char *anton):** this function updates a word synonym and antonym on the BST.
- **TTree *TraversalBSTinOrder(TTree *tr):** this function performs in-order traversal of the BST **tr**.
- **TTree *TraversalBSTpreOrder(TTree *tr):** this function performs pre-order traversal of the BST **tr**.
- **TTree *TraversalBSTpostOrder(TTree *tr):** this function performs post-order traversal of the BST **tr**.
- **void HighSizeBST(TTree *tr):** this procedure prints the high and size of the BST **tr**.
- **TTree *LowestCommonAncestor(TTree *tr, char *word1, char *word2):** this function returns the lowest common ancestor of two nodes (words).
- **Int CountNodesRanges(TTree *tr, int l, int h):** this function counts the number of nodes that lies within a given range [l, h].
- **TTree *inOrderSuccesor(TTree *tr , char *word):** this function returns the in-order successor of a given node in **tr**.
- **TTree *BSTMirror(TTree *tr):** this function returns a mirror of the BST **tr**.
- **bool isBalencedBST(TTree *tr):** this function checks if a given BTS **tr** is balanced.
- **TTree *BTSMerge(TTree *tr1, TTree *tr2):** this function merges two BST **tr1** and **tr2** into a single balanced BTS.

3.4. Modules based on Recursion (recursion.h/recursion.c) (08 points)

- **int countWordOccurence(File *f, char *word):** this function counts the number of occurrence of a word in the file **f** using recursion.
- **File *removeWordOccurence(File *f, char *word):** this function removes all occurrence of a **word** from the file **f** using recursion.
- **File *replaceWordOccurence(File *f, char *word, char *rep):** this recursive function replaces all occurrence of **word** from the file **f** by the string **rep**.
- **void wordPermutation(char *word):** this procedure prints all permutation of a given word using recursion.
- **void subseqWord(char *word):** this recursive procedure generates all possible subsequence of a given word.
- **int longestSubseqWord(char *word1, char *word2):** this recursive function finds the length of the longest common subsequence between two strings.
- **int distinctSubseqWord(char *word):** this recursive function counts the number of distinct subsequence of the given **word**.
- **bool isPalindromWord(char *word):** this recursive function checks if a given **word** is a palindrome.

3.5. Module 8: Complexity Analysis (complexity.c/complexity.h) (06 points)

- Measure Execution Time for different operations (Insertion, Deletion, Search).
- Compare Performance of Different Data Structures.
- Analyze Space Complexity for different implementations.

4. Module 9: Optional part (12 points)

Develop a graphical user interface (GUI) in C; for this aim, you can use libraries such as:

- GTK+: a popular library for cross-platform GUIs.
- ncurses: a text-based UI for terminal applications.
- Raylib: a simple library for graphical applications.

Academic Year: 2024/2025

Second Semester Project

Level: 1st Year **Material:** Algorithms and Dynamic Data Structures

- SDL (Simple DirectMedia Layer): a useful for rendering simple UI elements.

5. Report (40 points)

All the students are requested to submit their complete project using C programming language, libraries, test suite, and project report by May, 22nd 2025.

6. Appendix: Basic functions for files management

File Declaration:

```
File *file_name;
```

File opening:

```
file_name=fopen("input.txt", "rw"); // rw is the opening mode (read and write)
```

Read strings from a file line by line:

```
char buffer[100]; // buffer is a string where we put the entire line of the file
while (fgets(buffer, sizeof(buffer), file_name)!=NULL){
        //process on buffer where it is updated each time untill it is becomes null

}
```

Read formatted string from a file

```
char word[20];
char syn[20];
fscanf(file_name, "%[^=]=%[^\n]", word, syn);
```

or

```
// this is recommended to use inside the fgets loop bellow
if (sscanf(buffer, "%[^=]=%[^\n]", word, syn) == 2){
printf("word: %s, syn: %s\n", word, syn);
}
```

Write a string into a file

```
fputs(string, file_name); // write a string
```

or

```
fprintf(file_name, "%s#%s", string ) ; // write a formatted string
```

File close:

```
fclose(File_name);
```